

# From The rchitect

A Veeam Architect Paper

## Getting Started with Veeam REST APIs

Authors Edward Howard & Jorge De La Cruz

## Contents

Executive Summary .....	3
Target audience .....	3
Introduction .....	3
Understanding APIs.....	4
The world is API-driven .....	4
What creates responses?.....	5
Understanding HTTP requests .....	6
Response Codes .....	6
Authorisation .....	7
Basic Authentication .....	7
Self-Signed Certificates .....	8
OAuth 2.0 .....	8
Example 1 – Login and obtain a Bearer on Veeam Backup and Replication API .....	9
Working with Authorisation Response data .....	10
What are Arrays .....	10
What are Objects? .....	11
Practical Use.....	11
Using the API .....	13
Working with the Documentation .....	13
Making an Authorised GET request .....	15
Swagger UI .....	15
Swagger Login .....	16
API Credentials and Bearer .....	16
Job Sessions API call using Swagger .....	17
Postman .....	19
Making POST & PUT requests .....	21
Advanced manipulation techniques .....	21
Example – Using Jq to parse the Job Sessions JSON response .....	22
Example jq grabbing the name, endTime, result, and message from the last job .....	22
Conclusion.....	23
Appendix A – Python JSON Manipulation.....	24
Appendix B – Quick example using Bash Shell.....	25

## Executive Summary

Representational State Transfer Application Programming Interface or REST APIs provide a granular, flexible, and scalable method to report, manage, and automate Veeam applications. (REST API will be referred to as API for the remainder of this document).

This guide will outline how you use Veeam APIs in your data protection workflow with practical examples of how you can take advantage of this powerful resource.

## Target audience

This Whitepaper is mainly intended by a technical audience such as backup administrators and systems engineers. However, it also applies to anyone that requires a fast, efficient method of accessing and managing Veeam applications.

## Introduction

This Whitepaper doesn't assume any prior knowledge of command line or specific coding languages. However, to properly take advantage of the power of APIs, a coding language will be required.

This guide will show examples using PowerShell, Linux Shell, and Python. PowerShell and the Linux Shell can perform quick commands with little setup; however, data manipulation of the responses is arguably more tricky.

Curl with Jq and Python will be used for most of the examples in this Whitepaper as they have relatively simple syntax and are easier to learn.

Also, the principles that can be learned by using Curl and Python can be transferred to other languages such as PowerShell, JavaScript, C# and Go to and many others.

This guide will primarily focus on Veeam Backup and Replication; however, the principles that are shown can also be applied to:

- Veeam Enterprise Manager
- Veeam Service Provider Console
- Veeam Backup for O365
- Veeam Backup for AWS
- Veeam Backup for Azure
- Veeam Backup for GCP
- Veeam ONE

## Understanding APIs

The world of the web is underpinned by APIs or Representational State Transfer Application Programming Interface. It allows applications to communicate with each other with nothing more than a network or internet connection.

**Quick example:** Think of an API as a waitress, you have a menu, and you select some food, the waitress takes that order to the kitchen, who prepares the food, and the waitress delivers the order to your table:



## The world is API-driven

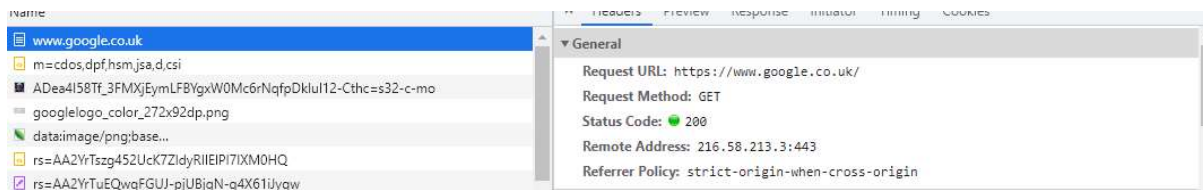
Do you remember the good old days when we were thinking about our holidays, searching for different Hotels, pricing, locations, transportation, activities to do at the destination, etc.? What a blast from the past, right? It never looked complex like going to 15 different websites to book everything, like this:



Instead, modern websites interact with each other, so the end-user, us in this case, could have a seamless experience. You can book the whole flight, transportation, hotel, activities, and payment, all without the need to go to different websites; in the background, what is happening is something like this. An application, or system, performs the required API calls, either GET, POST, etc. and builds a response to the end-user:



On a bit more technical note, and as a quick check: If you were to open your web browser and press CTL + SHIFT + I, then go to the Network tab, you could view some of these requests when you load a webpage.



Unlike CLI (Command Line Interfaces), APIs do not require a specific application to access them; the software needs to make an HTTP request.

### What creates responses?

An API Server is a web server that listens to incoming requests and, based on the information requested, sends back the requested data, usually in JSON (JavaScript Object Notation) format. It should be noted that XML (Extensible Markup Language) was commonly used before JSON.

JSON is easier to read and work with, taking over as the main data transfer formatting type.

The data within the JSON can either be hardcoded into files directly in the code itself or, more commonly, held in a Database. It is then translated into JSON format by the server application and sent back to the requester.

Note: The Veeam Enterprise Manager API defaults to XML response formatting. JSON needs to be specifically requested from the API to get a response in the formatting

```
"Content-Type": "application/json"
```

## Understanding HTTP requests

An HTTP request to an API requires different constructs depending on the requested action.

An HTTP request has several verbs that define the action that is being requested:

GET	Read data or resources
POST	Create data or resources
DELETE	Delete data or resources
PUT	Update data or resources

Source: MDN<sup>1</sup>

These are not the only options available, but these are the ones that are relevant for working with Veeam APIs.

GET requests are relatively easy and can be done using a simple command-line command.

### PowerShell

```
Invoke-WebRequest -URI https://jsonplaceholder.typicode.com/todos/1
```

### Curl

```
curl https://jsonplaceholder.typicode.com/todos/1
```

Note that Curl has been added to later versions of Windows, but it is a wrapper around Invoke-WebRequest. You can send Curl commands directly in Windows if you use either Gitbash<sup>2</sup> or WSL<sup>3</sup>.

POST and PUT requests tend to be more involved and usually require more setup to send the data to the API. These will be discussed in more length later in the Whitepaper.

## Response Codes

Response codes come back from the API in the response and provide quick information on how the API call went.

- **1xx: Informational** – Communicates transfer protocol-level information.
- **2xx: Success** – Indicates that the client's request was accepted successfully.
- **3xx: Redirection** – This indicates that the client must take additional action to complete their request.
- **4xx: Client Error** – This category of error status codes points the finger at clients.
- **5xx: Server Error** – The Server takes responsibility for these error status codes.

Reference: <https://restfulapi.net/http-status-codes/>

The key takeaway is that you want a **2xx** response code back from the API.

<sup>1</sup> <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>

<sup>2</sup> <https://gitforwindows.org/>

<sup>3</sup> <https://docs.microsoft.com/en-us/windows/wsl/install>

## Authorisation

Authorisation is arguably the most challenging part of using APIs, but it is a step that usually only needs to be done once. The great thing about coding is that it is easy to copy/paste code snippets from one program to another or even create reusable code in the form of modules that can be imported repeatedly. So, once you get past this step in your learning, things get a lot easier.

There are several different types of Authorisation available; however, for this Whitepaper, we will be focusing on two types which Veeam APIs use:

- Basic Auth
- OAuth 2.0

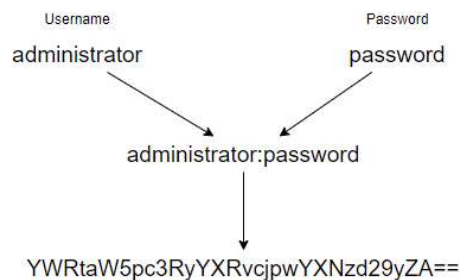
### Basic Authentication

As defined by the RFC 7617 standard, Basic Authentication transmits credentials as user-id/password pairs, encoded in base64.

Base64 is a method of encoding data; for example, "Hello World" would be encoded into "SGVsbG8gV29ybGQ=".<sup>4</sup>

Many languages will do the base64 encoding for you before sending the request; others may not. So, again you do need to check the specifics of your chosen language's HTTP package.

But for completeness, here is a representation of the conversation process:



With Basic Authentication, the credentials are held in the Header, so they need to be constructed into the header Object before sending to the API. No Body needs to be sent to the API as part of the authorisation process.

A Header can be seen as instructions that are being sent to the Server outside of the main data being sent in the Body. It is like an introduction in a Document.

If the Authentication is successful, a Token is sent back as part of the response header object; in Enterprise Manager's case, it is called "X-RestSvcSessionId". You can then use this Token in the Header of your requests to the API from that point, usually with a time-out that requires you to re-authenticate after a period of inactivity.

---

<sup>4</sup> <https://www.online-python.com/B8PhJU1WZ7> online example in Python

## Self-Signed Certificates

Many Veeam on-premises installations use self-signed certificates for the Veeam API, so the CA (Certificate Authority) cannot be verified. A signed certificate is highly recommended to apply to APIs, particularly web-facing APIs such as Veeam Backup for AWS/Azure/GCP, as it prevents "man-in-the-middle attacks". However, it is also possible to tell the HTTP software to ignore the certificate check. An example of this is Curl's `-k` or `--insecure` flag.

NOTE: Again, be very cautious of ignoring certificates.

## OAuth 2.0

OAuth 2.0 works differently and underpins all the newer APIs in the Veeam family of Products.

With OAuth 2.0, you need to send the Username and Password, but it is held in the request's Body instead of in the headers. It also requires you to specify a "Grant-Type" parameter when sending the request.

```
body = {
    "Grant-Type": "password",
    "Username": "administrator",
    "Password": "password"
}

headers = {
    "accept": "application/json",
    "x-api-version": "1.0-rev1",
    "Content-Type": "application/x-www-form-urlencoded"
}
```

You will note that the "Content-Type" parameter is "application/x-www-form-urlencoded" unlike with the Basic Auth which uses "application/json". As well as some other headers that are also required.

If you were to write out the body object yourself in x-www-form-urlencoded, it would need to look like this:

```
Grant-Type=password&Username=administrator&Password=password
```

However, thankfully, most languages again have either a method of "parsing" the data before being sent or doing the conversion automatically for you during the request—for example, Python's "Requests" library<sup>5</sup>.

The next step is sending a request to the Authorisation endpoint; an endpoint is a specific URL that handles a request type. For example, with the direct API from Veeam Backup and Replication, which was added in v11, the endpoint is:

```
https://your-vbr-address:9419/api/oauth2/token
```

If successful, like with Basic Auth, you will receive a "200" response code, and a "Bearer Token" will be included in the Body of the response.

---

<sup>5</sup> <https://2.python-requests.org/en/master/>



```
headers = {
    "accept": "application/json",
    "x-api-version": "1.0-rev1",
    "Authorization": "Bearer ..."
}
```

### Example 1 – Login and obtain a Bearer on Veeam Backup and Replication API

```
curl -X POST "https://YOURVBRIPORFQDN:9419/api/oauth2/token" -H "accept: application/json" -H "x-api-version: 1.0-rev1" -H "Content-Type: application/x-www-form-urlencoded" -d "grant_type=password&username=YOURUSER&password=YOURPASS&refresh_token=&code=&use_short_term_refresh="
```

1. We are using **Curl**, available natively on any Linux distribution
2. We are using **POST**, which means we are sending something to the Server (user and credentials in this case), hoping to obtain a response based on those
3. We use some Headers; in this case, we are using the **application/json**, the specific API revision we want, and finally, the **application/x-www-form-urlencoded**
4. We include the /d (the data we are sending over) to finish the API call, using basic OAuth 2.0. There you can see you need to add your user and pass

From here, you have a valid Auth Token that expires in 900 seconds (this time depends on the API):

[illegible]

## Working with Authorisation Response data

Response data can be simple data structures combined to make more complex ones.

This data is usually a mixture of two structures: an array known as a list and an object known as a dictionary. Different languages have different terms for these structures, but they do the same thing.

We will use the terms Array and Object for the remainder of this Whitepaper for ease.

## What are Arrays

Arrays are a method of holding several elements together, such as numbers, strings (text), or other Arrays and Objects, as we will see later.

Here is an example of an array of numbers:

```
myArray = [1, 2, 3, 4, 5]
```

Arrays are indexed, which means you can access an element but reference its location.

This would result in reference to the number 1 as arrays are zero-indexed.

Another example:

```
[{"item": 1}, {"item": 2}, {"item": 3}]
```

Again, this shows an array of objects to get the first value; you specify the index like before.

## What are Objects?

We have seen objects previously when creating headers and object "objects"; these look very similar to the JSON format. These are a series of key= value pairs separated by a colon. Multiple values can be created inside an object, including other objects.

```
myObject = {  
    "item": 1,  
    "info": {  
        "info1": 2,  
        "info2": 3  
    },  
}
```

This is an example of an object within another object; these are very common in responses.

You can also embed arrays in objects like so:

```
myObject = {  
    "item": 1,  
    "info": {  
        "info1": 2,  
        "info2": 3  
    },  
    "myArray": [1,2,3,4,5]  
}
```

To access the data within an array, you must provide the key to get the value. Using the example above to get the "item" value, you need to do one of the following.

```
myObject['item']  
myObject.item
```

## Practical Use

Earlier, it was mentioned that we needed to pull the Token out of the response to create the Header for future requests after authentication.

Using the techniques above that become relatively trivial, the examples below use Python syntax, but the general method is similar to other languages.

### Basic Auth

```
token = response['X-RestSvcSessionId']
```

Note that the key in Basic Auth has an invalid character for the "dot" notation shown in a previous example.

### OAuth

```
token = response.access_token  
token = response['access_token']
```

These techniques can construct the new Header required when sending future requests.

One method is to construct the new Header manually and add the correct key value.

## Basic Auth

```
token = response['X-RestSvcSessionId']  
  
new_header = {  
    "accept": "application/json",  
    "X-RestSvcSessionId": token  
}
```

## OAuth

```
token = response['access_token']  
  
new_header = {  
    "accept": "application/json",  
    "x-api-version": "1.0-rev1",  
    "Authorization": "Bearer " + token  
}
```

Various other formatting methods from different languages allow for a similar method of adding to an object.

Further techniques and tools will be discussed on pulling out relevant data from API responses later in the Whitepaper.

## Using the API

Now that we are ready to go, it is time to make an authorised request; however, we need to know what request we need to send before we do that.

### Working with the Documentation

One important aspect of using APIs is reading the Veeam documentation on constructing the request. The Veeam Help Center has extensive information on all the products that support APIs.

- <https://helpcenter.veeam.com/>

For example, you wish to request the Veeam Backup for Azure API to get a list of the restore points, so, on the Help Center, you will select Veeam Backup and Replication and select the REST API Reference:

All products > Veeam Help Center

Product:   
Veeam Backup & Replication

Version:   
11

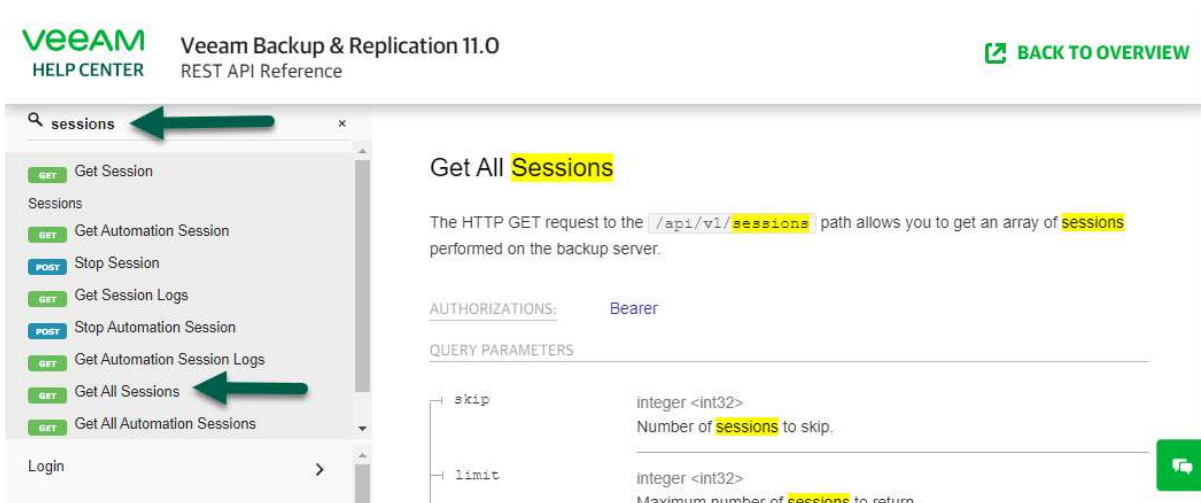
Type:   
All types

Language:   
English

### Veeam Backup & Replication

PRODUCT GUIDES		
Veeam PowerShell Reference	html	
Veeam Explorers PowerShell Reference	html	
Veeam Backup & Replication REST API Reference	html	
Veeam Backup Enterprise Manager REST API Reference	html	

Once we are on the REST API Reference, we can use the search to guide us. Imagine you are looking to get the Job Sessions:



Veeam Backup & Replication 11.0  
REST API Reference

BACK TO OVERVIEW

Search: sessions

- GET Get Session
- Sessions
- GET Get Automation Session
- POST Stop Session
- GET Get Session Logs
- POST Stop Automation Session
- GET Get Automation Session Logs
- GET Get All Sessions
- GET Get All Automation Sessions
- Login

### Get All Sessions

The HTTP GET request to the `/api/v1/sessions` path allows you to get an array of sessions performed on the backup server.

AUTHORIZATIONS: Bearer

QUERY PARAMETERS

skip	integer <int32> Number of sessions to skip.
limit	integer <int32> Maximum number of sessions to return.

The request format is shown as:

```
GET https://<hostname>/api/v1/sessions
```

It also provides options that can be added to the request as "query parameters" to specify specific items, a range of items, or filters based on various parameters.

For example, if you want to modify the maximum number of sessions, we can use the next as part of the URL.

```
https://<hostname>/api/v1/sessions?limit=10
```

The question mark defines the start of the query parameters, and these can be chained to create more refined requests by adding an ampersand.

```
https://<hostname>/api/v1/sessions?limit=10&typeFilter=Job
```

Each API document has a section called "Query Parameters", which details other options such as limiters, offsets, and search patterns. For example:

[https://helpcenter.veeam.com/docs/backup/vbr\\_rest/queries.html?ver=110](https://helpcenter.veeam.com/docs/backup/vbr_rest/queries.html?ver=110)

The Documentation also provides extensive information on the data that will be received back from the API, including types and an example of the structure of the response object. This is particularly important if you use strongly typed languages like C# or Golang.

Continuing using Curl as an example, sending a request to the API endpoint shown above will look like this:

```
curl -X GET "https://YOURVBRIP:9419/api/v1/sessions?limit=10&typeFilter=Job" -H "accept: application/json" -H "x-api-version: 1.0-rev1" -H "Authorization: Bearer eyJhbGciOiJSUzUxMiIsImtpZCI6IjM3RjI1MEMARdQ0NjhERDgyNkFFRkZCNDc4RjgwMERBQTZEEMEQ2QjciciLCJ0eXAiOiJKV1QiOyJ1bmllxdWVfYmFtZSI6IkpukDFREVMQUQSUVpcXGFkbWluaXN0cmF0b3IiLCJ0eXNyY0ie2NDQ0NDA2MDcsImV4cCI6MTY0NDg0MTUwNWwyawWF0iJjoxNjQ0ODQ0WnJA13LChZsdWoiOiJhY2NiLC3I0YyQyLTA3ZS8KbnBlc2VC5KhgyJT8qJHhzO4fl2-C75genKBnB74Fr18rVwTVcw8-JvP13GAKEG5ujfl2Y86aeNLfJVFS1DEXC9KA99XvkQ_SV7zBemAY2ktPtETKzEXoXSQQ1NN11x4qd9uLeIX7d94XLAsy6nrWZH00ibOyxDXz-hPFSAHZZZHx5f0mv-_wf7NsBCFOrSH_QexncxpJ_KolG81runsVTbmJLeM7tcwX2PkKkj9tqxW--d2A5NECuuiTcI9Pn2WmBdmHzihpyaAiEGYyd_TDROEmcTYDaEwmv3oSbB55yvyn5Cw--GQ7woN8l9qdeEbAcnu8UNblpKA"
```

Let's split this query a bit:

1. We are using **Curl**, available natively on any Linux distribution
2. We are using **GET**, which means we are expecting information if we are properly authenticated
3. We are using the VBR API IP, Port, the specific endpoint we want, with some filters
4. We use some Headers; in this case, we are using the application/json, and the specific API revision we want (this varies between the products)
5. We are trying to run this query using our Authentication Bearer; please refer to the Auth section if you are in doubt

The result of that specific endpoint will look something like this:

```
{
  "data": [
    {
      "sessionType": "Job",
      "state": "Stopped",
      "id": "b8fd1065-5caa-4e61-b301-e3122d200b89",
      "name": "VMware - Create NetApp Snapshot",
      "activityId": "3bad9a4f-efe3-4cd0-bfdc-7bf0a0c8cade",
      "creationTime": "2022-02-14T12:10:21.48+00:00",
      "endTime": "2022-02-14T12:10:30.013+00:00",
      "progressPercent": 100,
      "result": {
        "result": "Failed",
        "message": "Failed to create processing task for VM NGINX-004 Error: Cannot access VMX file of VM [NGINX-004]",
        "isCanceled": false
      },
      "resourceId": null,
      "resourceReference": null,
      "parentSessionId": null,
      "usn": 0
    },
  ],
}
```

The result is self-explanatory; it does include the **name of the Job**, the **endTime**, the **result**, an **additional message**, etc. It is usually an array with information so that we can grab the most important and useful data for us. We will take a quick example of how to parse the data later.

## Swagger UI

Getting our hands dirty using Curl, Python, or Bash is nice; in the end, we will need some programming language to build scripts that downloads data or automate and orchestrate things. But over the years, what we have found is that having a web portal where you can run the queries to obtain the data, do the auth, etc. It is more than enough for some Customers, plus the "baby first steps" that we all need.

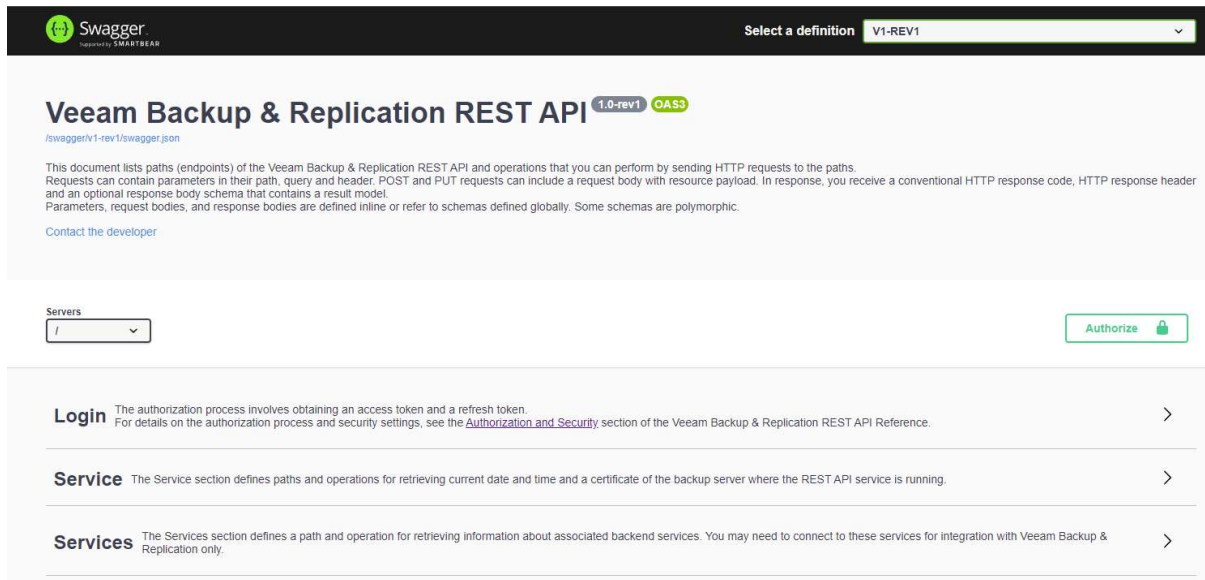
Swagger allows Veeam to describe the APIs' structure and wraps them up on a very simple and nice web interface that we can leverage to perform all the tasks.

Let's take the Veeam Backup and Replication API and use the Job Sessions example once again.

## Swagger Login

Every product has its own Swagger on different ports. We recommend looking into the Help Center, searching for the product you are trying to access, and getting the Swagger URL and port from there. On Veeam Backup and Replication, the Swagger can be found under:

- <https://VBRIP:9419/swagger/ui/index.html>



**Veeam Backup & Replication REST API** 1.0-rev1 OAS3

This document lists paths (endpoints) of the Veeam Backup & Replication REST API and operations that you can perform by sending HTTP requests to the paths. Requests can contain parameters in their path, query and header. POST and PUT requests can include a request body with resource payload. In response, you receive a conventional HTTP response code, HTTP response header and an optional response body schema that contains a result model. Parameters, request bodies, and response bodies are defined inline or refer to schemas defined globally. Some schemas are polymorphic.

[Contact the developer](#)

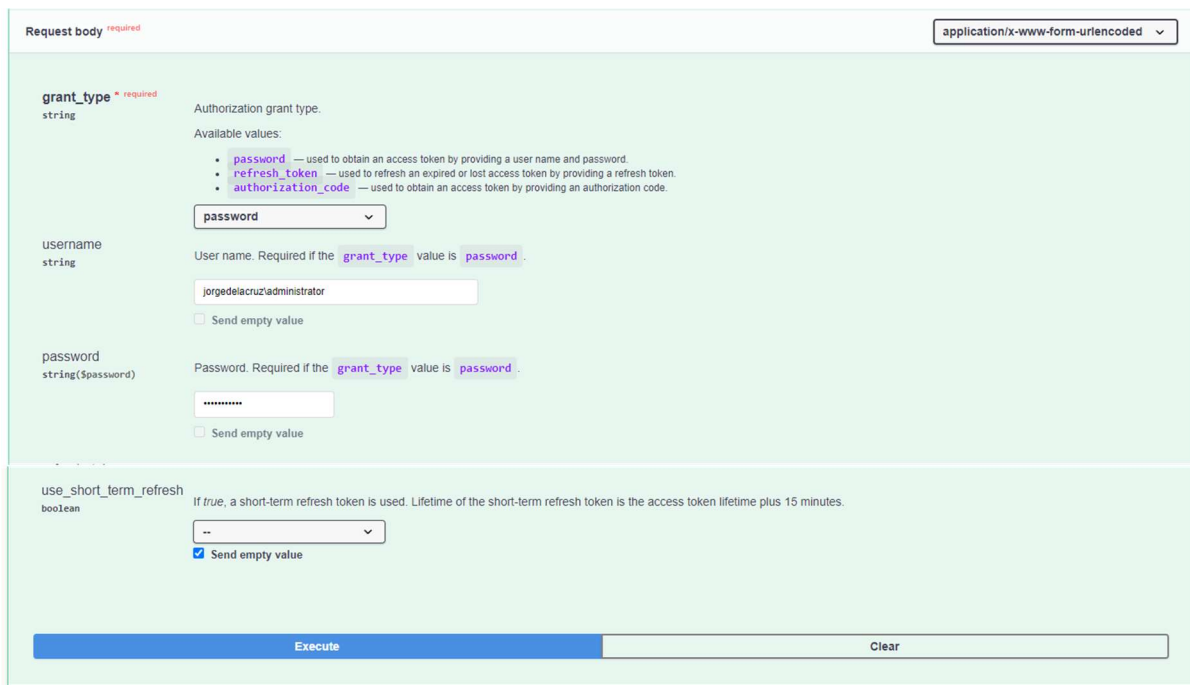
**Login** The authorization process involves obtaining an access token and a refresh token. For details on the authorization process and security settings, see the [Authorization and Security](#) section of the Veeam Backup & Replication REST API Reference.

**Service** The Service section defines paths and operations for retrieving current date and time and a certificate of the backup server where the REST API service is running.

**Services** The Services section defines a path and operation for retrieving information about associated backend services. You may need to connect to these services for integration with Veeam Backup & Replication only.

## API Credentials and Bearer

As mentioned before, we will always need a Token to perform any action inside the API; let's get our Token now. As simple as click under Login → Try it out → Introduce credentials → Execute



**Request body** required application/x-www-form-urlencoded

**grant\_type** \* required  
string  
Authorization grant type.  
Available values:  

- password** — used to obtain an access token by providing a user name and password.
- refresh\_token** — used to refresh an expired or lost access token by providing a refresh token.
- authorization\_code** — used to obtain an access token by providing an authorization code.

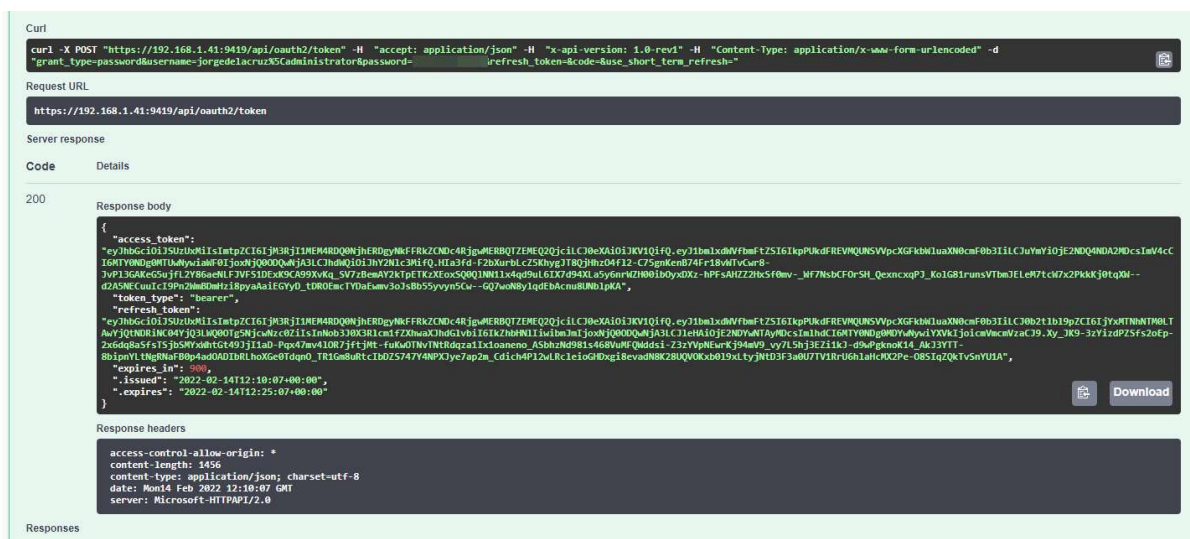
**username**  
string  
User name. Required if the **grant\_type** value is **password**.  
  
☐ Send empty value

**password**  
string(\$password)  
Password. Required if the **grant\_type** value is **password**.  
  
☐ Send empty value

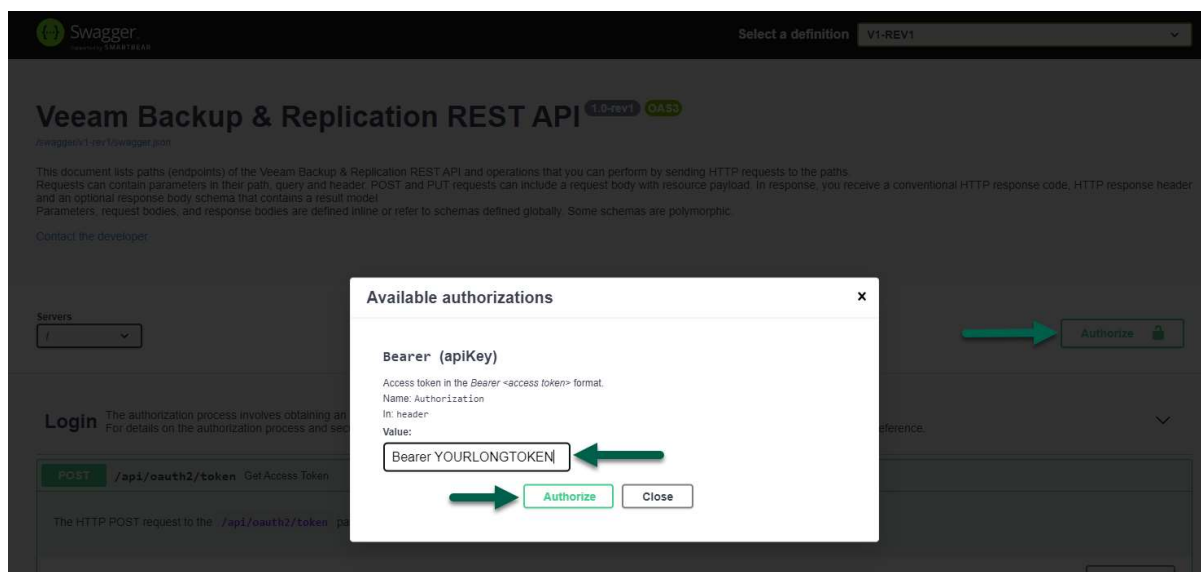
**use\_short\_term\_refresh**  
boolean  
If true, a short-term refresh token is used. Lifetime of the short-term refresh token is the access token lifetime plus 15 minutes.  
☐ ☒ Send empty value

**Execute** **Clear**

This will make the API call, and you will be able to see the Bearer token:

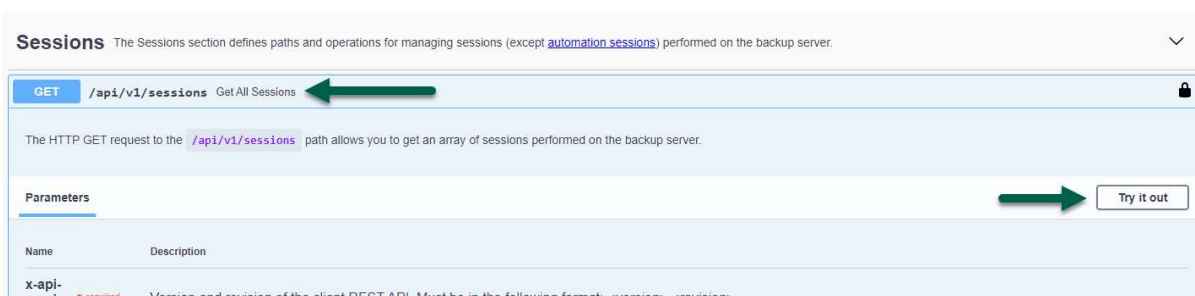


Let's copy the whole Token, inside the double quotes after access token; now we go to the top right where it says Authorize, a popup will open, on the text field, introduce Bearer YOURLONGTOKEN (the Token you have copied before)

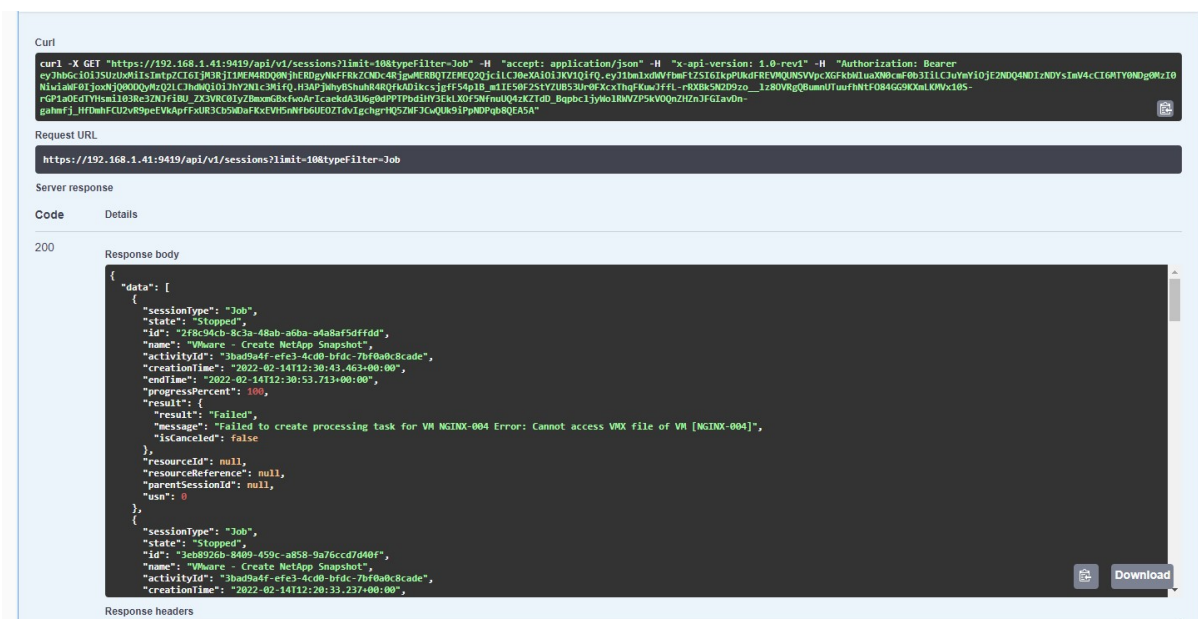


## Job Sessions API call using Swagger

We are almost there; once logged in, we can walk through the different endpoints, for example, the Job Sessions; let's click on Try it out, scroll till the end, and press Execute:



You will see the result of your query, right there, in JSON format:

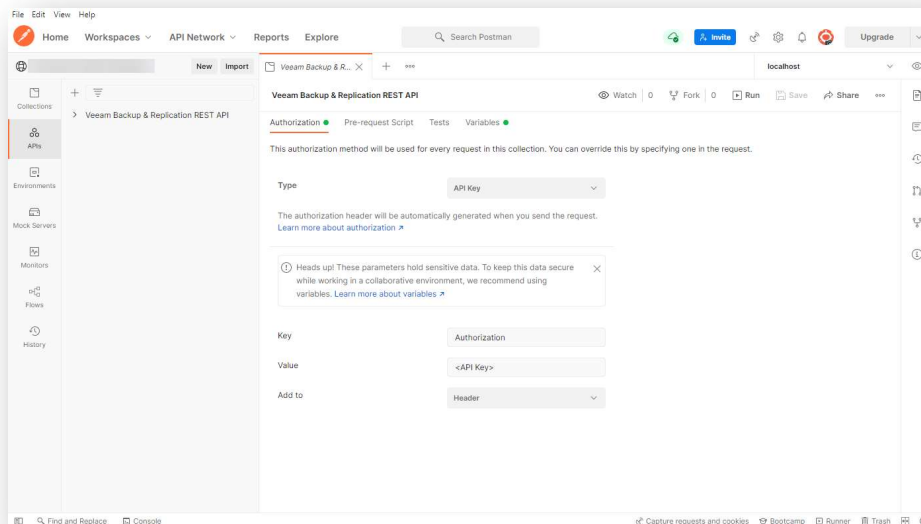


Swagger is a great way to explore the information, and from there, start building more complex applications, utilities, reports, etc.

## Postman

An alternative method of calling Veeam APIs is using Postman, which is available free from <https://www.postman.com/>

Postman makes it easy to send requests to APIs without code which can be used to learn the API's structure.



Veeam maintains a Postman collection of API calls that can be imported so you can quickly get going with APIs.

<https://github.com/VeeamHub/veeam-postman>

Download the files from VeeamHub, then click "import" on the appropriate Postman JSON file.

Some APIs require you to log into the Swagger interface and download the JSON file.

### Veeam Backup & Replication REST API

1.0-rev1 OAS3

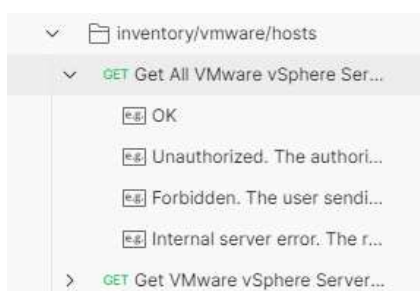
/swagger/v1-rev1/swagger.json

This document lists paths (endpoints) of the Veeam Backup & Replication REST API and operations that you can perform by sending HTTP requests to the paths. Requests can contain parameters in their path, query and header. POST and PUT requests can include a request body with resource payload. In response, you receive a conventional HTTP response code, HTTP response header and an optional response body schema that contains a result model. Parameters, request bodies, and response bodies are defined inline or refer to schemas defined globally. Some schemas are polymorphic.

Contact the developer

You can use this JSON file as if you were to import it from the downloaded Repository.

You will then see the various requests that can be sent listed.



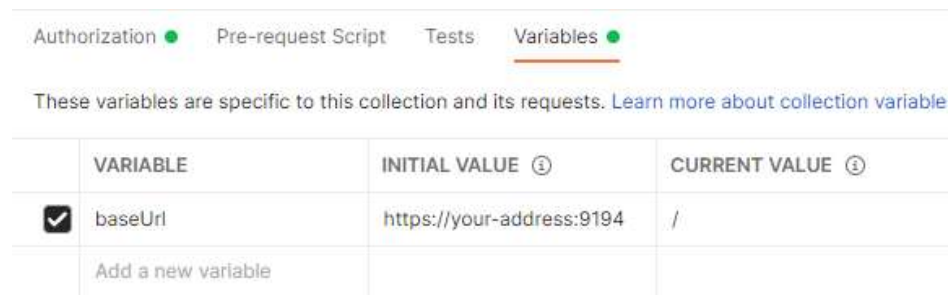
You will note that the URLs have double curly brackets around usually the beginning of the URL. These represent variables and makes it easier to upload all the calls in one go.



There are Global and Local variables, but usually, you will want to set the local variable.



Enter the start of the URL in the variable.



The Veeam Postman Collection is well documented and will help with working with the application.

## Making POST & PUT requests

GET requests are relatively simple as you only need to provide the information in the URL. POST requests require you to provide information in the Body of the request. A good example of this is with the Authorisation as seen previously.

Here are some examples of POST requests.

### PowerShell

```
$Body = @{
    title= 'Veeam API Whitepaper'
    body= 'Do Whitepaper'
}
Invoke-WebRequest -URI https://jsonplaceholder.typicode.com/posts -Body $Body -Method 'POST'
```

### Curl

```
curl --data' {"title": "Veeam API Whitepaper", "body": "Do Whitepaper"} 'https://jsonplaceholder.typicode.com/posts
```

Note: Jsonplaceholder is an example of a free open public API that allows you to test out using APIs.

In both commands, we add data to the post request in the "body", the normal location for the data, also known as the "payload".

There is also another structure called the "Request header", which provides information to the API about the request context. This can include what type of data you wish to receive from the API if there is a choice, e.g., Veeam Enterprise Manager.

The Headers are also often used to hold Authorisation data to allow the request to be serviced. This will be covered later in this Whitepaper.

Both the Body and the Headers are held in what is called a JSON string, which is a series of key-value pairs with outer curly brackets and the keys separated by the values using a colon.

## Advanced manipulation techniques

There are a few options to manipulate the data we obtain. The first and always recommended will be the Veeam Query Parameters we mentioned before, like skip, limit, Sort, Filter, etc. So, we reduce the pressure on the Server by just querying what is relevant.

Once we have the JSON response that we like, remember the Job Sessions one, we can quickly see that there is a lot of information that perhaps we do not need, or imagine if you want to grab some of that information and put it into variables.

To do this, we will use jq. Jq is like the Linux command sed; you can use it to slice, filter, map, or transform structured data in JSON format. Other languages have methods to do similar things.

Jq could be used online by pasting your JSON into, for example, [jqplay.org](https://jqplay.org). To understand a bit more about jq, please do visit their manual:

- <https://stedolan.github.io/jq/manual/>

This Whitepaper is not aimed to make you a Jq expert, but we will share some of the basic examples so you can get your head around it; from here, please ping us on the forums or social media.

### Example – Using Jq to parse the Job Sessions JSON response

Let's take as an example the Job Sessions. We have explained already what an array is, objects, etc. Quickly coming back into that part, we can see that this JSON has an array called data, and inside every group of objects are linked to a specific Job session:

```
{
  "data": [
    {
      "sessionType": "Job",
      "state": "Stopped",
      "id": "b8fd1065-5caa-4e61-b301-e3122d200b89",
      "name": "VMware - Create NetApp Snapshot",
      "activityId": "3bad9a4f-efe3-4cd0-bfdc-7bf0a0c8cade",
      "creationTime": "2022-02-14T12:10:21.48+00:00",
      "endTime": "2022-02-14T12:10:30.013+00:00",
      "progressPercent": 100,
      "result": {
        "result": "Failed",
        "message": "Failed to create processing task for VM NGINX-004 Error: Cannot access VMX file of VM [NGINX-004]",
        "isCanceled": false
      },
      "resourceId": null,
      "resourceReference": null,
      "parentSessionId": null,
      "usn": 0
    },
  ],
}
```

### Example jq grabbing the name, endTime, result, and message from the last job

Let's try to grab the information from the last job:

```
jq '.data[0] | .name, .endTime, .result.result, .result.message'
```

The result of this will be like this:

```
VMware - Create NetApp Snapshot
2022-02-14T12:30:53.713+00:00
Failed
Failed to create processing task for VM NGINX-004 Error: Cannot access VMX file of VM [NGINX-004]
```

As you can see, you have taken out all the JSON complexity and just grabbed everything you needed.

Let's write another example; imagine that the array is too long, and you only care for the jobs that start with "VMware ..." or any other string; you could use the select option together with jq, like this:

```
.data[] | select(.name | startswith("VMware"))
```

The result of this will give you all the information from the job sessions, that starts with VMware:

```
{
  "sessionType": "Job",
  "state": "Stopped",
  "id": "b8fd1065-5caa-4e61-b301-e3122d200b89",
  "name": "VMware - Create NetApp Snapshot",
  "activityId": "3bad9a4f-efe3-4cd0-bfdc-7bf0a0c8cade",
  "creationTime": "2022-02-14T12:10:21.48+00:00",
  "endTime": "2022-02-14T12:10:30.013+00:00",
  "progressPercent": 100,
  "result": {
    "result": "Failed",
    "message": "Failed to create processing task for VM NGINX-004 Error: Cannot access VMX file of VM [NGINX-004]",
    "isCanceled": false
  },
  "resourceId": null,
  "resourceReference": null,
  "parentSessionId": null,
  "usn": 0
}
```

There are many other ways to manipulate the data that comes back from an API, jq is just one of many, and each language has its own method of doing it.

To see the same process using Python, please see Appendix A.

## Conclusion

The use of APIs is an extremely powerful and flexible method of reporting and managing a Veeam backup environment.

It allows you to take full control of Veeam operations in the way that works for you, with whatever tool or codebase you want, and from anywhere there is connectivity. Do you want to connect using Golang in a K8s container or create a custom widget on your Desktop that pops an alert on a successful backup? It is all possible.

We hope that this Whitepaper has provided you with the initial basis of the benefits and use of Veeam APIs. It empowers you to see the potential of leveraging them in the future.

## Appendix A – Python JSON Manipulation

Using the same response as shown in the Advanced manipulation techniques section:

```
{
  "data": [
    {
      "sessionType": "Job",
      "state": "Stopped",
      "id": "b8fd1065-5caa-4e61-b301-e3122d200b89",
      "name": "VMware - Create NetApp Snapshot",
      "activityId": "3bad9a4f-efe3-4cd0-bfdc-7bf0a0c8cade",
      "creationTime": "2022-02-14T12:10:21.48+00:00",
      "endTime": "2022-02-14T12:10:30.013+00:00",
      "progressPercent": 100,
      "result": {
        "result": "Failed",
        "message": "Failed to create processing task for VM NGINX-004 Error: Cannot access VMX file of VM [NGINX-004]",
        "isCanceled": false
      },
      "resourceId": null,
      "resourceReference": null,
      "parentSessionId": null,
      "usn": 0
    }
  ],
}
```

To get a similar result as shown in the above section, we will use the "tabulate"<sup>6</sup> library:

```
from tabulate import tabulate

data = jobSessions['data']

headers = ["Name", "End Time", "Result", "Result Message"]

results = []

for i in data:
    results.append([i['name'], i['endTime'], i['result']['result'], i['result']['message']])

print(tabulate(results, headers=headers, tablefmt="grid"))

+-----+-----+-----+-----+
| Name           | End Time           | Result  | Result Message                                     |
+-----+-----+-----+-----+
| VMware - Create NetApp Snapshot | 2022-02-14T12:10:30.013+00:00 | Failed  | Failed to create processing task for VM NGINX-004 Error: Cannot access VMX file of VM [NGINX-004] |
+-----+-----+-----+-----+
```

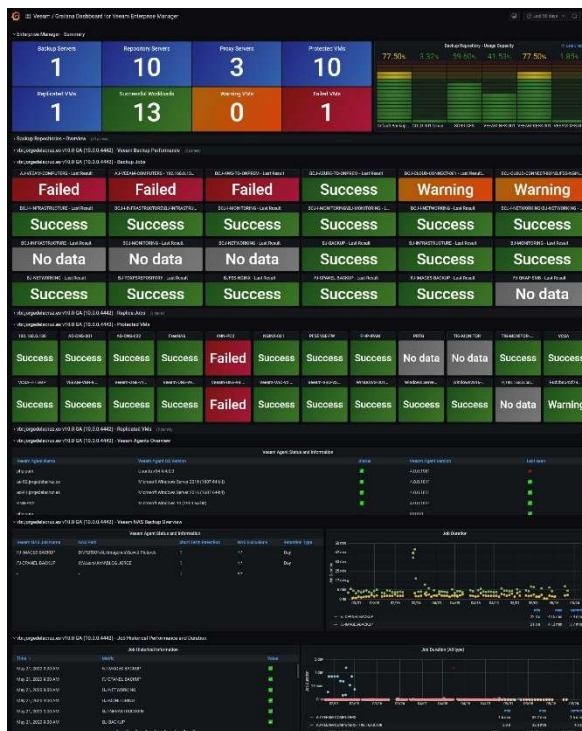
Note this assumes that the variable "jobSessions" holds the "data" object.

A loop allows for multiple entries to be displayed in the table.

<sup>6</sup> <https://pypi.org/project/tabulate/>

## Appendix B – Quick example using Bash Shell

Over the last few years, we have seen multiple projects emerging into the Veeam Hub that leverages all Veeam APIs and transforms those into beautiful [Grafana Dashboards](#) or useful HTML Reports.



Grafana Dashboard for Enterprise Manager 1

These examples have been built using Bash Shell Script, let's take a look at some parts of the Scripts, knowing now all the basics, we should be able to read a bit what they do:

Looking at the **Bash Shell Script** for Enterprise Manager, we can see at the top that is asking for some system variables like user, password, IPs, etc:

```
# Endpoint URL for login action
veeamUsername="YOU'REUSER"
veeamPassword='YOU'REPASSWORD'
veeamJobSessions="100"
veeamAuth=$(echo -ne "$veeamUsername:$veeamPassword" | base64);
veeamRestServer="YOU'RESERVERIP"
veeamRestPort="9398" #Default Port
veeamSessionId=$(curl -X POST "https://$veeamRestServer:$veeamRestPort/api/sessionMgr/?v=latest" -H
"Authorization:Basic $veeamAuth" -H "Content-Length: 0" -H "Accept: application/json" -k --silent | awk
'NR==1{sub(/\xef\xbb\xbf/, "")}' | jq --raw-output ".SessionId")
veeamXRestSvcSessionId=$(echo -ne "$veeamSessionId" | base64);

timestart=$(date --date="-1 days" +%FT%TZ)
```

EC2 Policy Backup: AWS EC2 - Snapshots and Backups				Success of VMs processed	
Friday, June 11, 2021 11:59:53 AM					
Start time			09:00:13 PM		
End time			09:06:25 PM		
Duration			06m:12s		
Details					
Name	Job Type	Status	Start time	Transferred	Duration
linumv03	EC2 Backup	Success	09:01:40 PM	19.5 MB transferred	0m:2m:55s
winumv01	EC2 Backup	Success	09:01:41 PM	81.1 MB transferred	0m:4m:43s
linumv01	EC2 Backup	Success	09:01:41 PM	19.6 MB transferred	0m:3m:55s
linumv02	EC2 Backup	Success	09:01:41 PM	19.8 MB transferred	0m:3m:34s
VMware Backup for AWS - Database:					

Veeam Backup for AWS - Hostname: https://3.64.76.69 Version: 3.1.0.19

EC2 Policy Snapshot: AWS EC2 - Snapshots and Backups				Success of VMs processed	
Friday, June 11, 2021 11:59:53 AM					
Start time				09:00:13 PM	
End time				09:01:40 PM	
Duration				0m:1m:26s	
Details					
Name	Job Type	Status	Start time	Transferred	Duration
linumv03	EC2 Snapshot	Success	09:00:31 PM	N/A	0m:1m:56s
winumv01	EC2 Snapshot	Success	09:00:33 PM	N/A	0m:1m:46s
linumv01	EC2 Snapshot	Success	09:00:34 PM	N/A	0m:1m:56s
linumv02	EC2 Snapshot	Success	09:00:34 PM	N/A	0m:1m:34s

Veeam Backup for AWS - Hostname: https://3.64.76.69 Version: 3.1.0.19

VPC Backup: VPC Configuration Backup				Success of VMs processed	
Friday, June 11, 2021 11:59:53 AM					
Start time		08:50:02 PM			
End time		08:50:02 PM			
Duration		0m:0m:19s			
Details					
Name	Job Type	Status	Start time	Transferred	Duration
EU Central (Frankfurt) - AWS Account 353812094947	VPC Backup	Success	08:00:06 PM	N/A	0m:0m:3s
Viewman Backup for AWS - Hostname: <a href="https://3.64.76.69">https://3.64.76.69</a> Version: 3.1.0.19					

Veeam Backup for AWS - Hostname: https://3.64.76.69 Version: 3.1.0.19

RDS Policy Snapshot: AWS SQL					Success of VMs processed
Friday, June 11, 2021 11:59:53 AM					
Start time		08:00:02 PM			
End time		08:00:23 PM			
Duration		0m:0m:19s			
Details					
Name	Job Type	Status	Start time	Transferred	Duration
aws01	RDS Snapshot	Success	08:00:06 PM	N/A	0m:0m:12s
View our backup for AWS - Hostname: <a href="https://13.64.76.60/Version: 1.1.0.19">https://13.64.76.60/Version: 1.1.0.19</a>					

Veeam Backup for AWS - Hostname: https://3.64.76.69 Version: 3.1.0.19

HTML Custom Report - Veeam Backup AWS 1

That was easy; it even manages the whole auth for us and puts the SessionID that the VEM API needs into a variable called `veeamXRestSvcSessionId`. Let's take a look at the next part; a few interesting parts here, like:

- We can see that we build first the full API URL we will query, in this case, the `/api/reports/summary/overview`
- Then the script does the curl query to the URL, using all the SessionID, and all the needed headers, and saves the output into another variable called `veeamEMOUrl`.
- Finally, using the magic of jq, we simply extract the objects we want and save them into variables, like `veeamBackupServers`, etc.
- The last step, in this specific case, is to echo the results on the console (on an InfluxDB format, meaning we can push those later on to InfluxDB, or any other Monitoring tool, by adjusting what the Monitoring tool expects)

```
##
# Veeam Enterprise Manager Overview. Overview of Backup Infrastructure and Job Status
##
veeamEMOUrl="https://$veeamRestServer:$veeamRestPort/api/reports/summary/overview"
veeamEMOUrl=$(curl -X GET "$veeamEMOUrl" -H "Accept:application/json" -H "X-RestSvcSessionId:$veeamXRestSvcSessionId" -H "Cookie: X-RestSvcSessionId=$veeamXRestSvcSessionId" -H "Content-Length: 0" 2>&1 -k --silent | awk 'NR==1{sub(/\xef\xbb\xbf/, "")}1')

veeamBackupServers=$(echo "$veeamEMOUrl" | jq --raw-output ".BackupServers")
veeamProxyServers=$(echo "$veeamEMOUrl" | jq --raw-output ".ProxyServers")
veeamRepositoryServers=$(echo "$veeamEMOUrl" | jq --raw-output ".RepositoryServers")
veeamRunningJobs=$(echo "$veeamEMOUrl" | jq --raw-output ".RunningJobs")
veeamScheduledJobs=$(echo "$veeamEMOUrl" | jq --raw-output ".ScheduledJobs")
veeamSuccessfulVmLastestStates=$(echo "$veeamEMOUrl" | jq --raw-output ".SuccessfulVmLastestStates")
veeamWarningVmLastestStates=$(echo "$veeamEMOUrl" | jq --raw-output ".WarningVmLastestStates")
veeamFailedVmLastestStates=$(echo "$veeamEMOUrl" | jq --raw-output ".FailedVmLastestStates")

echo "veeam_em_overview,host=$veeamRestServer
veeamBackupServers=$veeamBackupServers,veeamProxyServers=$veeamProxyServers,veeamRepositoryServers=$veeamRepositoryServers,veeamRunningJobs=$veeamRunningJobs,veeamScheduledJobs=$veeamScheduledJobs,veeamSuccessfulVmLastestStates=$veeamSuccessfulVmLastestStates,veeamWarningVmLastestStates=$veeamWarningVmLastestStates,veeamFailedVmLastestStates=$veeamFailedVmLastestStates"
```

We hope these examples are useful to you, and you could start leveraging all the Veeam APIs and adjusting the output to your requirements or tools.